Lab Assignment

CSE-245(Algorithm)

1. **Single-Source Shortest Paths – Dijkstra's Algorithm**
2. **Finding all the prime numbers less than N given number-The sieve of Eratosthenes algorithm**

Submitted To :

Jesan Ahammed Ovi

B.Sc. and M.S. in CSE, University of Dhaka.

Senior Lecturer, Dept. of CSE, East West University.

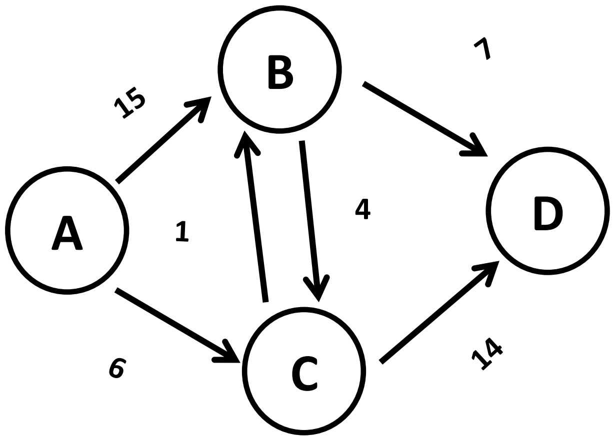
Submitted by:

Sajidur Rahman

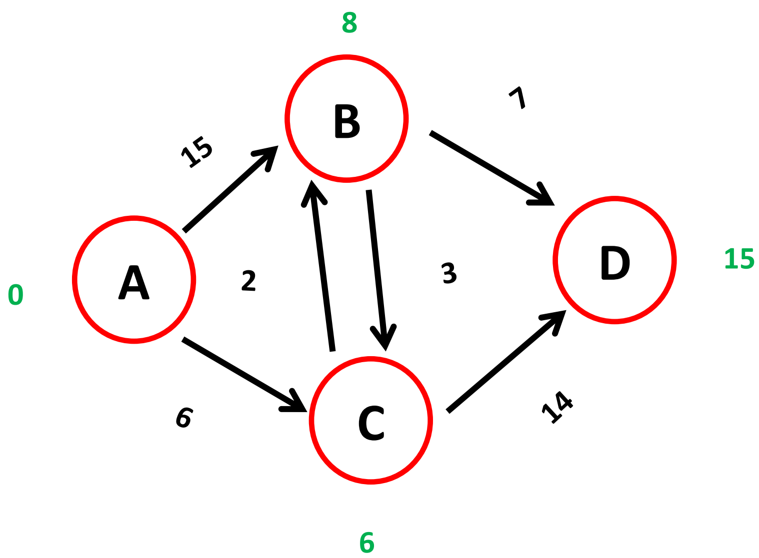
Id:2018-1-60-261

1. Single-Source Shortest Paths – Dijkstra's Algorithm

Problem statement: **Given a source vertex s from a set of vertices V in a weighted digraph where all its edge weights w(u, v) are non-negative, find the shortest path weights d(s, v) from source s for all vertices v present in the graph.**

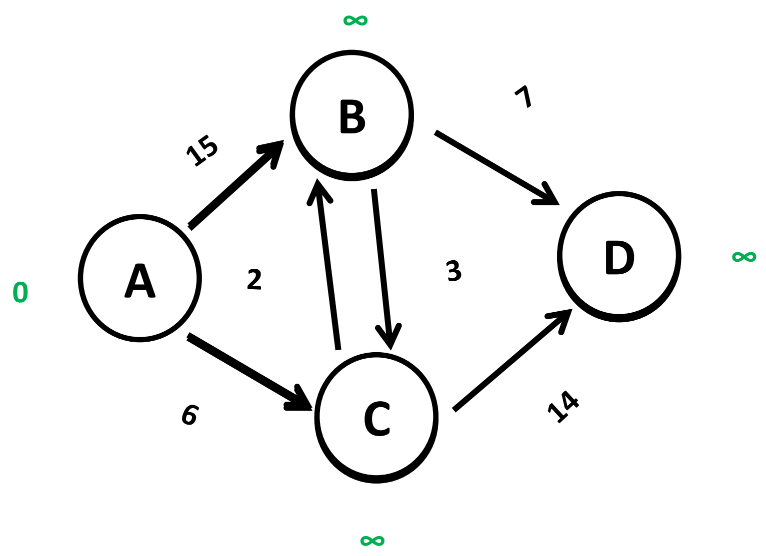


Vertex	Minimum Cost	path
A —> B	8	A -- > C --> B
A —> C	6	A --> C
A —> D	15	A --> C --> B --> D

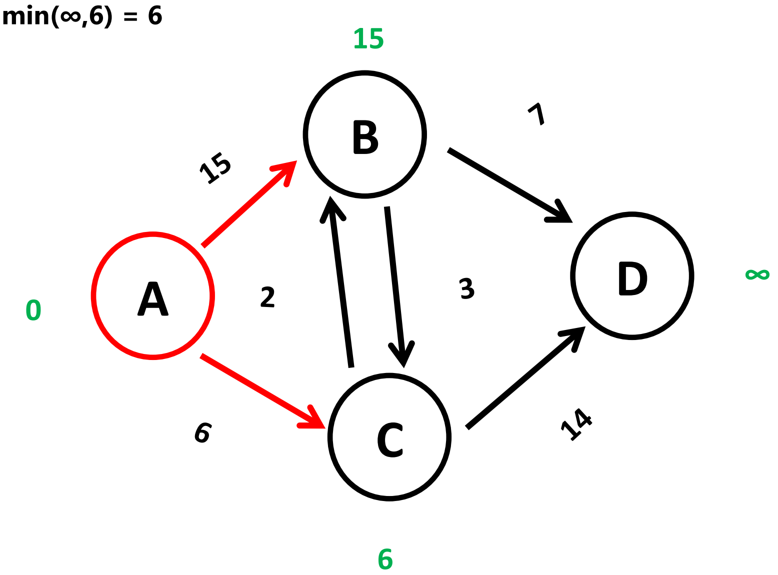


consider the following graph. We will start with vertex A, So vertex A has a distance 0, and the remaining vertices have an undefined (infinite) distance from the source. Let S be the set of vertices whose shortest path distances from the source are already calculated.

Initially, S contains the source vertex. S = {A}.



We start from source vertex A and start relaxing A's neighbours. Since vertex B can be reached from a direct edge from vertex A, update its distance to 15 (weight of edge A–B). Similarly, we can reach vertex C through a direct edge from A, so we update its distance from INFINITY to 6.
Min cost of A->B = min(∞,15) = 15
Min cost of A->C = min(∞,6) = 6
S={A}



After processing all outgoing edges of A, we next consider a vertex having minimum distance. B has a distance of 15, C has distance 6, and the remaining vertices D have distance INFINITY. So, we choose C and push it into set S.

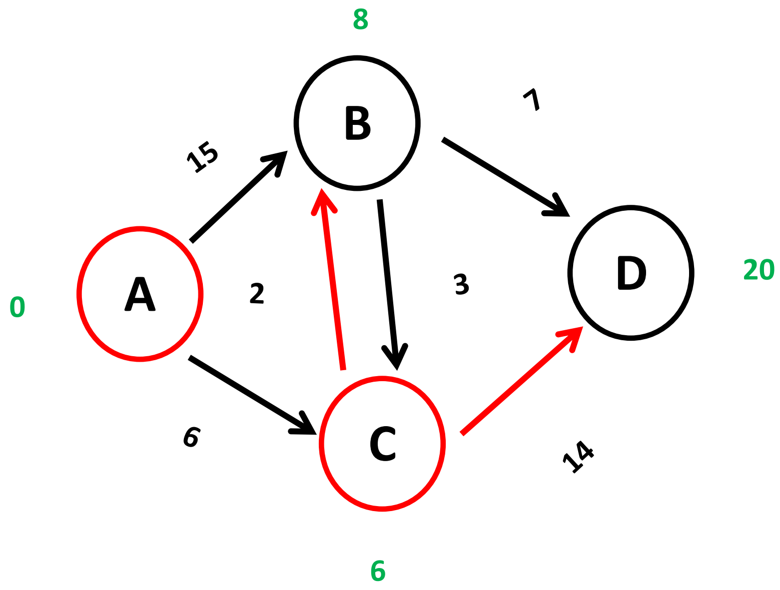
cost of A->B = A->C+C->B = 6+2 = 8

cost of A->D = A->C+C->D = 6+14 = 20

Minimum cost of A->B = min(15,8) = 8

Minimum cost of A->D = min(∞,20) = 20

S={A,C}



we choose B and push it into set S. B's neighbours are D and C.

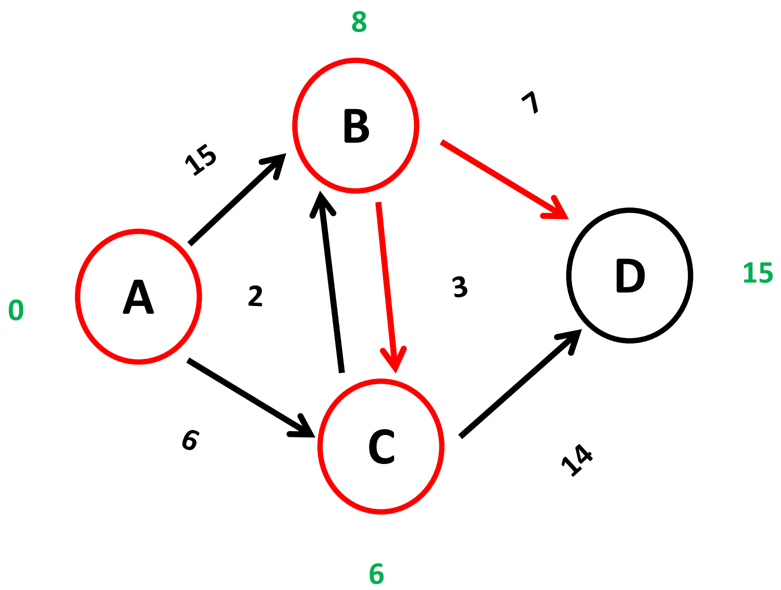
cost of A->C = A->B+B->C = 8+3 = 11

cost of A->D = A->B+B->D = 8+7 = 15

Minimum cost of A->C = min(6,11) = 6

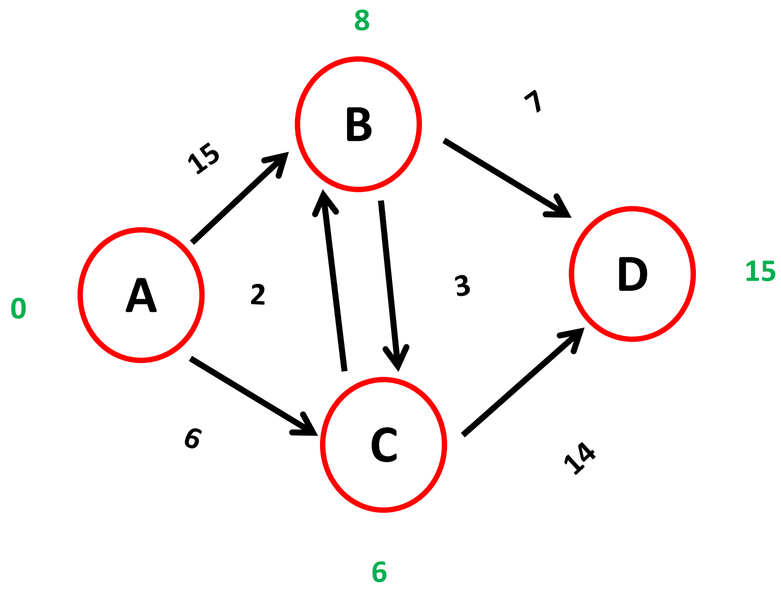
Minimum cost of A->D = min(20,15) = 15

S = {A,C,B}



we choose D and push it into set S. As, D has no Outgoing edges . D has no neighbours to relax.

S = { A,C, B,D}



PSUDO CODE:

```
function Dijkstra(Graph, source)

   dist[source] = 0    // Initialization
   create vertex set Q

   for each vertex v in Graph
   {
      if v != source
      {
         dist[v] = INFINITY       // Unknown distance from source to v
         prev[v] = UNDEFINED       // Predecessor of v
      }
      Q.add_with_priority(v, dist[v])
   }
   while Q is not empty
   {
      u = Q.extract_min()         // Remove minimum
      for each neighbor v of u that is still in Q
      {
         alt = dist[u] + length(u, v)
         if alt < dist[v]
         {
            dist[v] = alt
            prev[v] = u
            Q.decrease_priority(v, alt)
         }
      }
   }
   return dist[], prev[]
```

SOURCE CODE:

```
#include <bits/stdc++.h>

using namespace std;

struct Edge {
  int source, dest, weight;
};

struct Node {
  int vertex, weight;
};

class Graph {
  public:

    vector < vector < Edge >> adjList;

  Graph(vector < Edge >
    const & edges, int n) {

    adjList.resize(n);

    for (Edge
      const & edge: edges) {

      adjList[edge.source].push_back(edge);
    }
  }
};

void printPath(vector < int >
  const & prev, int i, int source) {
  if (i < 0) {
    return;
  }
  printPath(prev, prev[i], source);
  if (i != source) {
    cout << ", ";
  }
  cout << i;
}

struct comp {
  bool operator()(const Node & left,
    const Node & right) const {
    return left.weight > right.weight;
  }
};
```

```
//Dijkstra's algorithm
void findShortestPaths(Graph
  const & graph, int source, int n) {

  priority_queue < Node, vector < Node > , comp > min_heap;
  min_heap.push({
    source,
    0
  });

  // set initial distance from the source to v as infinity
  vector < int > dist(n, INT_MAX);

    dist[source] = 0;

  vector < bool > done(n, false);
  done[source] = true;

  // stores predecessor of a vertex to a print path
  vector < int > prev(n, -1);

  while (!min_heap.empty()) {

    Node node = min_heap.top();
    min_heap.pop();

        int u = node.vertex;

    for (auto i: graph.adjList[u]) {
      int v = i.dest;
      int weight = i.weight;

      if (!done[v] && (dist[u] + weight) < dist[v]) {
        dist[v] = dist[u] + weight;
        prev[v] = u;
        min_heap.push({
          v,
          dist[v]
        });
      }
    }

    done[u] = true;
  }

  for (int s = 0; s < n; s++) {

    if (s != source && dist[s] != INT_MAX) {
      cout << "Path (" << source << "->" << s << "): Minimum cost = "
<<
        dist[s] << ", Route = [";
      printPath(prev, s, source);
      cout << "]" << endl;
    }
  }
}
```

```
int main() {

  vector < Edge > edges = {
    {0,1,15},
    {0,2,6},
    {1,2,3},
    {1,3,7},
    {2,1,2},
    {2,3,14}};

  int n = 4;

  Graph graph(edges, n);

  // run the Dijkstra's algorithm from source node
  int source = 0;
  findShortestPaths(graph, source, n);

  return 0;
}
```

OUTPUT:

```
Path (0->1): Minimum cost = 8, Route = [0, 2, 1]
Path (0->2): Minimum cost = 6, Route = [0, 2]
Path (0->3): Minimum cost = 15, Route = [0, 2, 1, 3]

Process returned 0 (0x0)   execution time : 1.641 s
F]ress any key to continue.
```

Time Complexity:

Here ,Time complexity is O(E.log(V)). Here, E is the total number of edges, and V is the graph's number of vertices.